An Analysis of Vulnerability Trends, 2008 - 2016

D. Richard Kuhn[1], M S Raunak[2], Raghu Kacker[1]
kuhn@nist.gov, raghu.kacker@nist.gov raunak@loyola.edu
[1]National Institute of Standards and Technology [2]Loyola University of Maryland

Computer security has been a subject of serious study for at least 40 years, and a steady stream of innovations has improved our ability to protect networks and applications. But attackers have adapted and changed methods over the years as well. Where do we stand today in the battle between attackers and defenders? Are attackers gaining ground, as it often seems when reading press accounts of the latest data exposure? This analysis seeks to answer these questions using data from the US National Vulnerability Database (NVD) [1], and to identify classes of vulnerabilities where improvements will be most cost effective.

Data. The NVD is the US government's repository of information system security vulnerabilities. It is operated by the US National Institute of Standards and Technology, and is sponsored by the Department of Homeland Security's National Cyber Security Division. The NVD relies on publicly reported vulnerabilities from the Common Vulnerabilities and Exposures (CVE) dictionary. As of Spring 2017, there are more than 83000 vulnerabilities enumerated in the database. The NVD adopted Version 2.0 of the Common Vulnerability Scoring System (CVSS) in June 2007 to score the severity of each reported vulnerability, prior to the period in which this analysis begins. To ensure maximum consistency of data scoring and definition, we have used only reports from the period 2008 to 2016.

Vulnerability Severity. One area in which some progress is apparent is in the severity of vulnerabilities that are being discovered. For the NVD, severity is rated using the CVSS, which combines scores for impact and exploitability. As can be seen in Table I and Fig. 1, the proportion of high severity vulnerabilities is trending downward, declining about 15 percentage points since 2008. About two-thirds of this fraction has shifted to Medium severity vulnerabilities, which increased from about 46% to 55% of the total, while Low severity numbers increased from 3% to nearly 10% of the total.

Vulnerability Types. Table II shows the primary vulnerability categories used in the NVD. Each reported CVE is assigned to one or more categories called the Common Weakness Enumeration (CWE). Some of these primary CWE categories may include a number of subsidiary weaknesses. For example, CWE-119, Buffer errors, includes 14 subsidiary CWEs, such as out of bounds read (CWE-125), and untrusted pointer dereference (CWE-822). NVD entries in the 2008 to 2016 period were categorized as one of these types, with the exception of some which could not be determined because of insufficient information.

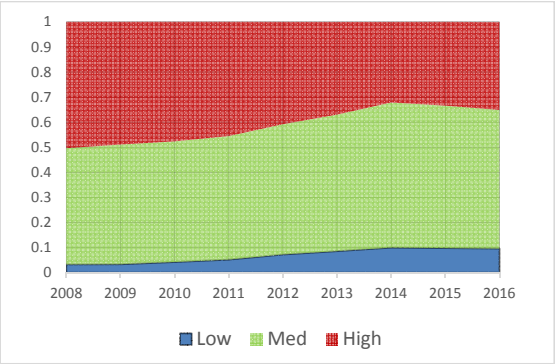

Fig. 1. Vulnerability Severity Trends, 2008-2016

TABLE I. VULNERABILITY SEVERITY, 2008-2016

	Low	Med	High
2008	0.033	0.463	0.504
2009	0.034	0.477	0.489
2010	0.043	0.481	0.477
2011	0.052	0.493	0.455
2012	0.073	0.519	0.408
2013	0.086	0.544	0.369
2014	0.100	0.579	0.322
2015	0.098	0.568	0.334
2016	0.096	0.553	0.350

TABLE II. NVD VULNERABILITY CATEGORIES

CWE-ID	Description	Type	Trend
CWE-16	Configuration	C	↓
CWE-20	Input Validation	I	↑
CWE-22	Path Traversal	I	↓
CWE-59	Link Following	I	≈
CWE-78	OS Command Injections	I	↑
CWE-79	Cross-Site Scripting (XSS)	I	≈
CWE-89	SQL Injection	I	↓
CWE-94	Code Injection	I	↓
CWE-119	Buffer Errors	I	↑
CWE-134	Format String Vulnerability	I	≈
CWE-189	Numeric Errors	I	↓
CWE-200	Information Leak / Disclosure	C	↑
CWE-255	Credentials Management	D	↑
CWE-264	Permissions, Privileges, Access	D	↑
CWE-287	Authentication Issues	D	≈
CWE-310	Cryptographic Issues	D	↑
CWE-352	Cross-Site Request Forgery	I	≈
CWE-362	Race Conditions	I	↑
CWE-399	Resource Management Errors	I	↓

We grouped the NVD CWE classes into primary types of Configuration, Design, and Implementation errors, designated in Table II as C, D, and I respectively. Table II also indicates whether the different vulnerability types are increasing (↑), decreasing (↓), or approximately unchanged (≈). In determining the type of each CWE class, we considered the common errors in each type. Configuration vulnerabilities result when a system is not set up correctly with respect to security goals. A simple example would be failure to enable password checking. Information leak is a broader type, but in most cases,

available security controls have been neglected or set up improperly, so this is designated as a Configuration error. Design-related vulnerabilities are those that originate in the planning and design of the system, such as selecting an outdated or weak cryptographic algorithm. The third source of vulnerabilities is typically simpler, but may have dramatic results. One of the most common implementation vulnerabilities is the simple buffer overflow. Failure to check that input size is within maximum buffer size is a simple error that should almost never occur, but continues to be a widespread problem (Table III). Some categories are less obvious. For instance cross-site scripting can have several forms, but in each case results from missing or inadequate input validation, so this is also included in implementation errors. Most of the other implementation-related vulnerabilities in Table II also result from failure to properly validate input.

What is most striking about the distribution of Configuration, Design, and Implementation errors captured in Fig. 2 is that implementation or coding errors account for roughly two thirds of the total. We consider the proportion of implementation vulnerabilities, rather than absolute numbers, because the number of vulnerabilities is partially a function of the number of applications released, which has increased over time. The proportion of implementation vulnerabilities for 2008 to 2016 is close to the 64% reported for 1998 to 2003 in an analysis of an early version of NVD [2]. This suggests that little progress has been made in reducing these vulnerabilities that result from simple mistakes which should be easy to prevent.

But this also means there is potential for significant reductions in vulnerabilities. Clearly better testing could prevent most such simple errors from making it into a released product, and practices such as code reviews and static analysis checks can be especially cost-effective for simple errors. Static analysis has been shown to detect about 20% of CVE-defined errors [3], and formal code inspection may prevent an average of about 65% of errors from reaching released products [4]. Thus vulnerabilities could be reduced with broader use of such practices.

To see the potential for improving cybersecurity through basic development practice, consider the absolute numbers of vulnerabilities shown in Table III (cryptographic issues adjusted for a spuriously large number in 2014 due to multiple entries resulting from failure to check X.509 certificates in Android apps). Implementation errors are highlighted in bold type; they represent a total of 27 242 of the 37 325 categorized vulnerabilities, or 72.9% for the 2008-2016 period. Note in particular that two of the presumably simplest errors to prevent, basic input validation and buffer errors, account for more than a third of the implementation flaws.

While the basic recommendations in this paper, greater use of static analysis tools and code review, have been made many times in the past [2], we note that progress has been made in static analysis, notably in the reduction of false positives and improved detection [3] [5], and code review is consistently shown to be highly cost effective [6]. This analysis will be extended to review trends within the different vulnerability types and subsidiary weaknesses, with a goal of identifying practices that may have the strongest impact on reducing vulnerabilities.

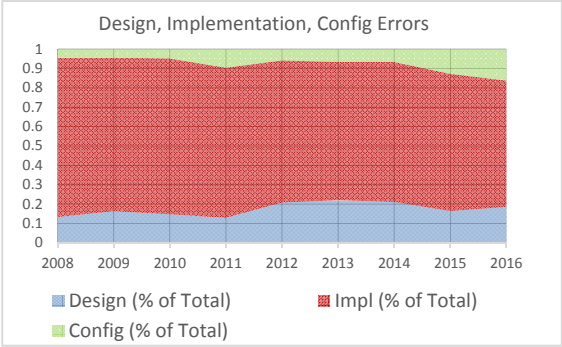

Fig. 2. Vulnerability Class Trends, 2008-2016

TABLE III. VULNERABILITY COUNTS FOR 2008-2016

Format String Vulnerability	**110**
Configuration	195
OS Command Injections	**208**
Race Conditions	**377**
Link Following	**389**
Credentials Management	589
Cryptographic Issues	779
Authentication Issues	920
Cross-Site Request Forgery (CSRF)	**1161**
Numeric Errors	**1199**
Code Injection	**1545**
Path Traversal	**1686**
Information Leak / Disclosure	2939
Input Validation	**3763**
SQL Injection	**3828**
Permissions, Privileges, and Access	4661
Cross-Site Scripting (XSS)	**6220**
Buffer Errors	**6756**
Total	37325

Products may be identified in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology or the US Government, nor that the products identified are necessarily the best available for the purpose.

[1] National Vulnerability Database, http://nvd.nist.gov 2017

[2] Heffley, Jon, and Pascal Meunier. "Can source code auditing software identify common vulnerabilities and be used to evaluate software security?" *System Sciences, 37th Annual Hawaii Intl Conf*, IEEE, 2004.

[3] Okun, Vadim, Aurelien Delaitre, and Paul E. Black. "Report on the static analysis tool exposition (SATE) IV" *NIST Special Publication* 500 (2013): 297.

[4] Jones, C, "Measuring Defect Potentials and Defect Removal Efficiency", *Crosstalk, Journal of Defense Software Engineering* (June 2008).

[5] Medeiros, I., Neves, N. and Correia, M., 2016. Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Trans. Reliability*, *65*(1), pp.54-69.

[6] Balachandran, V., 2013, May. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Software Engineering (ICSE), 2013 35th International Conference on* (pp. 931-940). IEEE.